Aequitas: A Trusted P2P System for Paid Content Delivery

Alex Sherman, Japinder Chawla, Jason Nieh, Cliff Stein, and Justin Sarma
Department of Computer Science
Columbia University
{asherman, jsc2127, nieh, cliff, jns2111}@cs.columbia.edu

Abstract

P2P file-sharing has been recognized as a powerful and efficient distribution model due to its ability to leverage users' upload bandwidth. However, companies that sell digital content on-line are hesitant to rely on P2P models for paid content distribution due to the free file-sharing inherent in P2P models.

In this paper we present Aequitas, a P2P system in which users share paid content anonymously via a layer of intermediate nodes. We argue that with the extra anonymity in Aequitas, vendors could leverage P2P bandwidth while effectively maintaining the same level of trust towards their customers as in traditional models of paid content distribution. As a result, a content provider could reduce its infrastructure costs and subsequently lower the costs for the end-users.

The intermediate nodes are incentivized to contribute their bandwidth via electronic micropayments. We also introduce techniques that prevent the intermediate nodes from learning the content of the files they help transmit.

In this paper we present the design of our system, an analysis of its properties and an implementation and experimental evaluation. We quantify the value of the intermediate nodes, both in terms of efficiency and their effect on anonoymity. We argue in support of the economic and technological merits of the system.

1 Introduction

P2P file-sharing has been recognized as powerful and efficient distribution model due to its ability to leverage users' upload bandwidth. (Popular examples include BitTorrent [5], Napster [24] and Kazaa [21]). However, companies that sell digital content on-line such as Apple Itunes [3], Sony, and Time Warner among others typically rely on "direct download" methods where the users download content either from the vendor's website directly or via a contracted CDN (content delivery network) such as Akamai [2]. Content providers are hesitant to rely on P2P systems for paid content distribution as "free-of-charge" file sharing is inherent in current P2P models.

This concern on behalf of the content providers is well-justified as P2P users easily form free file-sharing communities such as Xbox-sky [35] and Red Skunk Tracker [33]. Some systems such as Prodigem [27] require payment before a user can enter the P2P system. However, once the user is in the system, she directly shares the purchased content with other users via BitTorrent. Once these users learn of one another and know they have similar interests they can easily form a private P2P community for free future sharing of similar content.

In this paper we present Aequitas, a P2P system in which users pay for a file and then share parts of the file without ever learning of one another's identities. Instead, the users share the file via an additional set of intermediate P2P nodes that themselves are oblivious to the data that they help transmit. Since the users download data from these oblivious nodes, rather then their file-sharing peers, the system creates significant barriers against collusion for future free file-sharing. Thus, with Aequitas the vendors could leverage P2P bandwidth while effectively maintaining the same level of trust towards their customers as in a "direct download" system. The content providers could reduce the costs of their bandwidth and download infrastructure. Lower infrastructure costs translate into lower content costs for the end-users.

We envision a system with thousands of P2P nodes. For each file-sharing instance intermediate nodes are selected from this pool to transmit data between the file-sharing users. To incentivize these intermediate nodes (that we also call *Inodes*) to contribute their bandwidth we use electronic micropayments. These economic incentives come in terms of actual cash as opposed to credit for future resource usage as in many P2P systems. The cash payments are well-grounded in the profit excess recognized by the media vendors.

Finally, each Inode is restricted (via certificate checking) to downloading/uploading only a random fraction of the given file. While this restriction still allows these nodes to transmit parts of the file between file-sharing users, it prevents Inodes from obtaining the actual file without payment.

1.1 Properties and Side-Effects

We now state the main properties and side-effects of our system. The properties are proven in detail in section 4.

- **Anonymity.** Becuase we use intermediate nodes, Users in a given file-sharing instance are highly unlikely to learn of one another's identities and collaborate in a way that may hurt the content provider's interests. For instance, they are unlikely to leverage the system to form a private P2P sharing forum in the future or expose the IPs of the current file-sharing users and allow unauthorized users to free-load.

- **Authorized Download.** Since the Inodes only transmit randomly selected parts or (*chunks*) of the file we show with high probability that no small set of nodes can collude to obtain the copy of the file without payment.

- **Small Load on Content Provider.** We show that although the content provider is used in the system to fill out some missing chunks of the file for each user, the total load on the content provider imposed by the system is small (less than 1%).

- **Fair Compensation.** We guarantee that each node is paid at least the agreed amount for its bandwidth. This property simply stems from the feature of the system where each node first cashes the electronic coins for its services before uploading the piece of data.

In addition to the main properties Aequitas carries a number of beneficial side-effects that set it apart from many other P2P systems: privacy, fairness and node-longevity.

- **Privacy** In most P2P systems, such as BitTorrent, a spying node can simply request a particular file and easily learn the IPs of a number of machines that are also sharing this file. In our system, such node would only talk directly to Inodes and thus would fail to learn of other users.

- **Longevity and Fairness.** In most P2P system users prefer to close their clients as soon as they finish downloading what they need. In Aequitas, where all nodes are fairly compensated for the use of their bandwidth the users have an incentive to leave their software running longer. The longer nodes continue to run the more aggregate bandwidth is available to others for file downloads, making the downloads more efficient.

1.2 The Economic Model

Aequitas provides a way for the vendors to cut their costs by leveraging P2P bandwidth. However, P2P nodes must be incentivized and compensated for the use of their bandwidth with micropayments. We argue that the model that we present is a reasonable economic model for both the vendors and the participating P2P nodes.

Market research [1] suggests that digital media vendors spend 20% of their revenue on infrastructure costs for serving content. In theory, as long as the micropayments paid to the P2P nodes in total are less than the 20% than the vendors spend, the system should be a win for the vendors. Also, as long as the micropayments are greater than 0 it should be a win for the participants who otherwise waste their unused bandwidth.

Consider a vendor similar to Apple Itunes that sells videos thar range between 20 and 200 MBytes for $1.99. Assume that the average file size is 100MB, and that the payments to the P2P nodes are between the above thresholds of 0 and 20%, say 15%, or 30 cents per video. That is some collection of P2P nodes in total is compensates 30 cents when one user downloads a file.

If Aequitas uses two Inodes to transmit content between each pair of sharing users then to download each chunk requires upload bandwidth of three nodes. So if an average file size is 100MB, to download the entire file we actually use 300MB (100MBx3) of P2P bandwidth. The payment is 30 cents total to all the nodes that aid in one file download. Thus a single node would get 1 cent per 10MB of its upload bandwidth. If we further assume that a typical broadband participant has 300Kbits/sec of upload bandwidth to contribute, then when fully utilized this participant will be making 10.8 cents/hour or $78 /month! Even at a fraction of full utilization it is beneficial for a user to join Aequitas rather then loose their unused upload bandwidth. (Note, that to more fully utilize their bandwidth we allow a node to serve as an Inode in multiple file-sharing instances at once).

In the rest of the paper we describe the design and evaluation of our system. We show how the system adheres to the properties stated above. In evaluation we show that with extra layers the system does not add performance overhead to the traditional "direct download" model.

We present the related work in section 2. We describe the architecture and the mechanisms in section 3. We analyze the properties of the system in section 4. In section 5 we briefly describe some implementation features and detail evaluation results. We conclude in section 6.

2 Related Work

2.1 P2P File-Sharing

The earliest P2P file-sharing systems include CAN [28], Chord [32], Pastry [13] and Tapestry [37]. These systems use a variant of Distributed Hash Tables (DHT) to route messages and lookup objects and nodes. Unlike these systems, Aequitas is not a pure P2P system and uses externally managed services for routing and electronic banking. Aequitas does use Consistent Hashing [20] (often used in DHTs) but only to ensure the consistency of intermediate node assignments and not for consistent object location.

A number of public file-sharing systems like Kazaa [21], Napster [24] and Gnutella [18] have attracted much attention in recent years as these systems were often use to share copyrighted material illegally. BitTorrent [5], another public P2P system splits a large file into multiple chunks, thereby allowing a user to downloads different chunks from multiple peers simultaneously. This novel approach has been shown to exhibit higher fault-tolerance and to have more overall network efficiency, as each user explores multiple paths. At its core Aequitas is closest to BitTorrent as it also splits the files into a number of chunks. The main techinical distinctions are that Aequitas introduces anonymous sharing via intermediate nodes, and users pay each other with cash for chunk downloads. Aequitas and BitTorrent also differ in their intended uses: Aequitas create a system for content providers to distribute paid content rather while BitTorrent facilitates illegal sharing.

2.2 Anonymity

There has been much work in anonymity for network communications. MixNets [6] were introduced in 1981 primarily for anonymous email communication. To hide communicating nodes MixNets used sets of mixed paths, encryption, variable delays and message re-ordering. More recently this research has been followed by works like MixMaster [22], Mixminion [17], and MorphMix [30] that use similar techniques to maximize the anonymity in today's Internet.

Another system, GAP [4], does not focus on hiding communicating end-points. Rather it enforces anonymity by making it hard for an adversary to distinguish between an initiator of a message and an intermediary handling the message. Crowds [29] uses a large "crowd" of nodes to provide anonymity for the sender of a message.

Many of the anonymous systems were motivated by the Internet censorship and malicious traffic tracing. One simple system called Anonymizer implements a web proxy that strips the users information to allow anonymous browsing. Onion Routing [9] and its second generation Tor [11] allowed nodes to construct more complex routing circuits where each node in the path only has the knowledge of its successor and predecessor. There have also been P2P approaches to anonymity such as Tarzan [16]. In these systems the intermediate nodes are selected at random from a large set of P2P nodes. Tarzan allows a sender to construct a random path to a destination node encapsulating each message in a public-key encryption layer for each intermediate node in the path. Tarzan was specifically intended as a low-level anonymity layer for various network applications including browsing and file-sharing. I2P [19] is another mechanism that is designed to provide an anonymous layer for P2P applications.

Aequitas also selects intermediate nodes from a large P2P network. However, in contrast to the prior work the intent of anonymity in Aequitas is to anonymize the file-sharing Peers from one another, and make their direct communication difficult and unlikely.

Among systems that use both anonymity and P2P file distribution are Freenet [7] and GhostShare [23]. Both of these systems anonymize the requestor of the data. GhostShare also anonymizes the publisher, while Freenet anonymize the storage nodes. However these schemes, unlike Aequitas, do not attempt to hide the data from unauthorized users. In addition, GhostShare and Freenet do not guaranteed to store data for any length of time.

2.3 Economic Incentives

Many people have looked at the problem of encouraging efficient resource-sharing and discouraging free-ridership in P2P systems.

Some systems such as Samsara [8] recommend token exchange to encourage fair resource sharing. BitTorrent [5] itself is a token-based system. It implements a tit-for-tat protocol that encourages fair data exchange between users.

Some anonymity P2P systems encourage participation through reputation-building [10, 12, 12].

Systems like Karma [34] put a value on resources shared that allows participants to build credit and used resources against that credit.

Finally, systems like [15, 14] suggest electronic micropayments [31, 36, 26] for service of anonymous commnunication. Aequitas also uses electronic micropayments, but is different in that nodes pay for bandwidth, while anonymity is guarateed via controlled mapping. Systems that use cash provide much clearer incentives for participants then token or credit-building. The reason micropayments work so well with Aequitas is that the payments are grounded in the profits realized by the content providers. It is the content provider that distributes cash for users to pay one another, rather then individual users paying out of their pockets.

3 System Architecture

3.1 Overview

In Aequitas users (or *Peers*) share parts of the file via a swarming technique similar to BitTorrent's. The file is split into *chunks* that can be downloaded independently and in parallel from different Peers. The difference between Aequitas and BitTorrent is that the Peers in Aequitas are not aware of one another. They discover chunks and download them from one another exclusively via Inodes.

Before any Peers can request a file, the Content Provider or *CP* initializes one or more *seeds* with the entire file. ("Seed" is a term borrowed from Bit-Torrent and denotes a Peer that contains all chunks of the given file). A Peer joins a file-sharing instance by "buying" the file from the *Bank* and getting a *purchase certificate*. In addition, the Peer receives electronic cash that it will use to pay for the download service from the Inodes. As discussed in section 1.2 the value of this electronic cash is only a fraction of the cost of the file (e.g. 15%).

Next the Peer contacts the DTS (distributed tracking service) that checks the purchase certificate of the Peer and gives it a set of Inodes to talk to. The Peer uses these Inodes to discover and download the chunks. Typically a Peer will contact DTS only once and use the same set of Inodes for the entire download. The only reason to come back to DTS would be if many of its Inodes die or disappear. The Inodes assigned to the Peer will also need to contact DTS to obtain their respective sets of Inodes or Peers to forward requests to.

Towards the end of the download it is likely that the Peer will not be able to get a small fraction (less than 1%) of the chunks The Peer completes the download of missing chunks directly from the Content Provider.

It is interesting to mention that unlike pure P2P systems such as Chord, Aequitas manages services such as DTS and Bank separately from the P2P nodes. The reasons for these are simple - allowing P2P nodes to perform their own Peer discovery instead of the DTS would defeat the purpose of anonymity. The Bank is also responsible for P2P billing and

therefore cannot be implemented by the nodes themselves. Furthermore, it is the bandwidth that is constraining resource in the content delivery system, and it is precisely this resource for which we rely on the content-sharing Peers.

In the following sections we discuss individual components and heuristics used in the system.

3.2 The Bank

The Bank is the entity that maintains the credit for each user and provides the electronic cash service. The bank is owned and operated by the Content Provider (or CP). Once a month the CP sends a bill or check to the users based on their balance with the Bank. The users interact with the Bank in two important ways. First they pay the Bank when they wish to purchase a content file. Second they can cash electronic coins at the Bank. The electronic coins signed by the Bank are used by the users to pay for download services to one another. Nodes that provide services can cash the coins that they receive at the Bank and have their accounts credited.

Figure 2 demonstrates these two types of interactions with the Bank. At first Peer A contacts the Bank to purchase the content and obtain electronic coins that it will use to pay for all future downloads of the individual chunks. At some later point Peer A transmits some coins to a Node B and asks it for a chunk. Node B then attempts to cash the coins with the Bank. The Bank verifies its own signature on the coins and if valid adds credit to B's account. A coin Id included with the certificate prevent multiple cashing of the same coin. Upon successful cash operation node B sends the data to Peer A.

3.2.1 Micropayments

In Aequitas it costs one electronic coin to download a fixed size chunk of data from another node. As described above a node A that wishes to download data from node B first sends the required number of coins to node B.

In figure 1, for instance, node A can download chunks from node D via a path of length 3. A needs 3 coins to pay all of the nodes on the path. To download a chunk node A begins by transmitting 3 coins to node B. Node B cashes one of them and transmits 2 coins to node C. Node C cashes one and sends the remaining coin to node D. Node D cashes the last of 3 coins and sends the chunk to node C. The chunk is then transmitted down the path to node A.

In order for the Peer to have enough coins to pay for the download of all the chunks we need to account for the length of the *download path* (i.e. the number of edges that a chunk travels between Peers). For example, if the paths are of lengths 3 as in figure 1 and the number of chunks is 1000, the Bank gives a Peer 3000 coins upon content purchase.

3.2.2 Block Requests

Since cashing requires interaction with the Bank it creates a protocol overhead. To minimize this overhead, nodes request chunks in blocks and submit a corresponding number

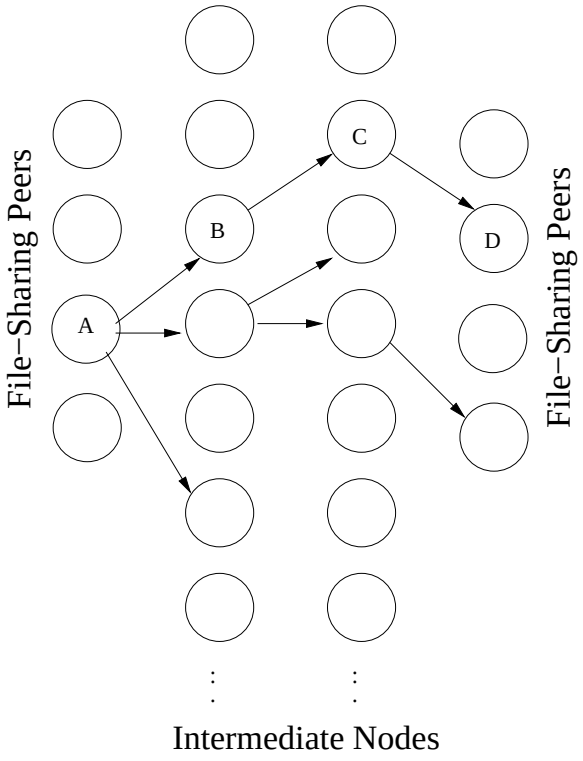

Figure 1: In this setup nodes communicate with nodes in adjacent columns. Nodes in columns 1 and 4 are file-sharing peers that share pieces of a file via intermediate nodes in column 2 and 3.

of coins at a time. The bigger the request, the smaller the overhead.

However, making requests too big exposes one's risk to coin loss. A mis-behaving node (whether it's malicious or malfunctioning) may cash the coins and not return the data. This creates a trade-off for picking the block size.

The loss of a fraction of coins is not catastrophic. The CP simply forgives the debt to the Peer and allows them to complete the download of any missing chunks in the end. However, to reduce the risk of malicious nodes stealing lots of coins we use a reputation heuristic discussed in section 3.5.

3.3 DTS

DTS tracks the set of available P2P nodes by listening to liveness pings from them. It also keeps track of all the Peers and Inodes that belong to a given file-sharing instance. Finally, the DTS hands out *node-maps* and *chunk-maps* to the nodes. These *maps* guide the nodes during the download process.

Both Peers and Inodes receive *node-maps*, a set of nodes that they go through to request file chunks. By handing out node-maps DTS implicitly builds a download graph similar to figure 1. In addition, Inodes receive *chunk-maps* that list the set of randomly selected chunks that a given Inode is allowed to serve. This restricted set of chunks guarantees that an Inode or even a large set of Inodes will not be able to ob-

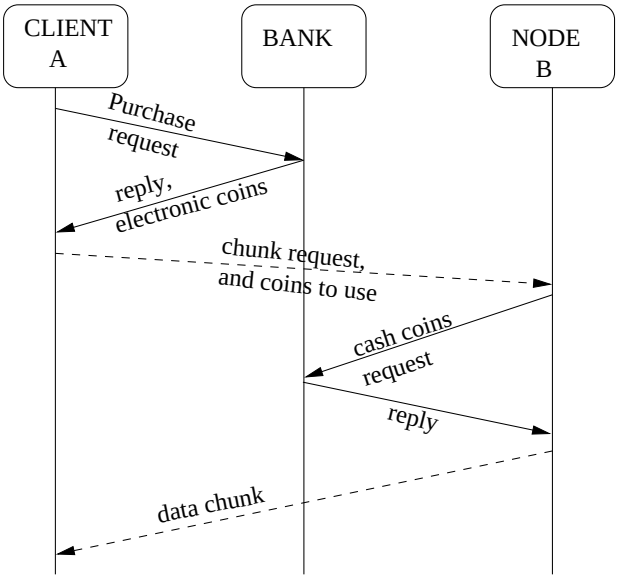

Figure 2: Bank Operation. First Peer A purchases the content and obtains coins. Second Node B receives coins from Peer A, and cashes them at the bank.

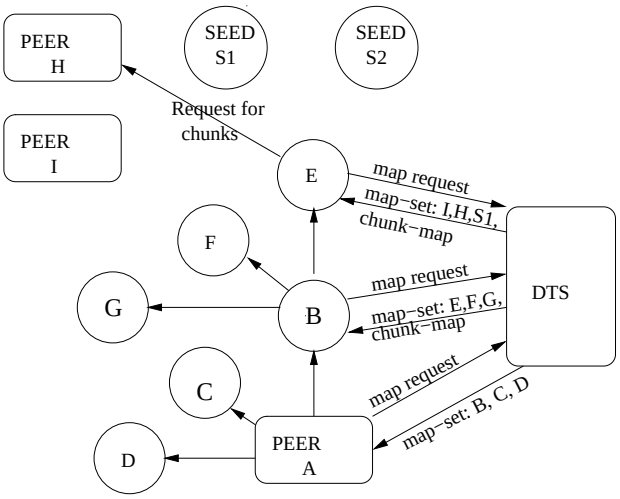

Figure 3: Interaction of Peers and Inodes with DTS.

tain a full copy of the shared file. The nodes request the maps before their first download request for chunks of a given file. The maps are certificates signed by the DTS. These certificates must be presented together with the future download requests.

Figure 3 demonstrates the use of DTS during the download process. When Peer A enters the system, and before it makes a first download request it contacts the DTS. It receives a *node-map* consisting of nodes B, C and D. When it first requests content through node B, node B contacts DTS and receives its map-set consisting of E, F and G. Node B also receives a chunk-map. When Node E receives a first request for the given file (from node B) it contacts the DTS and is mapped to some Peer H.

3.3.1 Node-Maps

By handing out node-maps the DTS implicitly constructs a download graph similar to figure 1 for each file-sharing instance. In figure 1 Peers in the left-most column download chunks from the Peers in the right-most column via two intermediate Inodes. We call the Inodes in the two middle columns *I1s* and *I2s*. We now go through the logic of how such a graph is constructed.

First in order to both send and receive data all Peers are replicated in both the left-most and the right-most column, meaning each Peer is mapped to both I1s and I2s. Second we note that to best protect anonymity of the Peers we try to minimize the number of Peers that each Inode has contact with. Thus we try to construct the mapping such that each I1 and I2 is in direct contact with very few Peers. (In a large P2P system where we select Inodes from a very large set we can guarantee with high probability that following our construction each Inode will be in contact with exactly one Peer.)

We construct the node-maps in the following way. We set a parameter D which is the out-degree of the Peers in the left-most column. When a new Peer joins the file-sharing instance we pick exactly D I1 and D I2 Inodes from the pool of available P2P nodes using consistent hashing. The node-map of the Peer contains the D I1s that it is hashed to. The node-map given to the I2 nodes is just the reverse of the consistent hash of the Peer (i.e. the Peer itself). Each I1 Inode is mapped to D of the existing I2 nodes also via consistent hashing.

The reason we use consistent hashing to map Peers is because it has a nice property that the result of the hash remains roughly unchanged despite nodes joining and leaving the system. This property helps keep the total number of Inodes talking to each Peer to a minimum. Thus if k I1 nodes initially given to a Peer die or become unavailable, and a Peer asks DTS for a new mapping, it will get $D - k$ original nodes plus k new nodes.

Picking the value of D involves a trade-off. On the one hand we want D to be small so that the number of Inodes in direct contact with a Peer is small. On the other hand larger D offers more parallelism and link fault-tolerance during the download. Also, since each Inode is restricted to downloading only a fraction of chunks, the more nodes a Peer talks to the higher is the total fraction of chunks that it can get through all of its Inodes. We discuss in section 4 the typical scenario for $D = 20$.

Of course we envision a system where the number of participants is large and we could add $2D$ Inodes for each new Peer. However, in our evaluation where we were limited in the number of total nodes we use a smaller D. Because of the smaller then ideal size of the network it is also possible in our evaluation that the Inodes are in contact with more than one Peer, although we keep that number small.

3.3.2 The number of Inodes

Throughout the paper we discuss the system where the downloads take place through a set of two Inodes. We justify the use of two Inodes in the following way. If we were to use only one Inode between Peers, then each Inode would be in touch with at least two file-sharing Peers and it would be easier for malicious nodes to break the anonymity of the Peers. When the number of Inodes is bigger than one each Inode can be constrained to talk to only one Peer. A malicious Inode would then require other colluding nodes to break the anonymity of some Peers. This requirement creates a much higher threshold for malicious behavior.

Of course, using more than two Inodes only enhances the anonymity of the system. However, the longer the path the higher the cost of the payments for the use of the Inodes' bandwidth.

3.3.3 Chunk-Maps

Chunk-maps handed out to Inodes contain a bitmap of the chunks of a given file that the Inode is allowed to serve. The chunk-map is determined with a deterministic hash function that takes as input the system id of the node, the filename and the fraction of the chunks to be assigned. The purpose of the chunk-maps is to prevent an Inode or set of Inodes that help transmit the file from obtaining the full file without paying for it. Since the chunks are assigned by a random deterministic function by the Coupon Collector's principal it is easy to see that with N chunks it would take at least $log(N)$ Inodes to collect the entire file. In section 4 we show using Chernoff Bounds that for appropriately chosen parameters a large set of Inodes will not obtain the full file with high probability.

3.3.4 Map Certificates

Both the node-maps and chunk-maps are presented as certificates signed by the DTS. When node A establishes a TCP connection with node B it presents these certificates. Node B verifies these certificates once for each TCP connection. It checks that for the given file node A is allowed to make requests to B and is allowed to ask for the requested chunks. Violations of the protocol are reported to the DTS, and the reputation heuristic discussed in section 3.5 removes the violating nodes from the system.

The overhead of certificate checking is quite small as it's done only once per TCP connection. A TCP connection is used to download all the future chunks of a given file that node A will request from node B.

3.3.5 Multiple Content Versions

It is possible that an Inode selected to help transmit a file will later want to purchase that file. The problem with that is that this Inode is already in direct communication with other Peers of that content. If we change it from being an Inode to being a Peer in the same file-sharing instance we break the anonymity constraint between sharing Peers.

To deal with this problem the content provider always creates several qualitatively similar versions of each content, but with different watermarkings.

For each file we split all available P2P nodes into several pools corresponding to the number of versions of that file. Each node can be assigned as an Inode to help distribute all but one of the versions of this file. If it purchases the file, that is the version that it gets.

The filename that the nodes exchange during chunk requests are one-way hashes of the actual content names. Thus an "Inode turned Peer" will not be able to tell that it served as an Inode for similar content either from the filename or from the binary file representation.

3.3.6 Scalability

The DTS service consists of several machines for fault-tolerance and scalability reasons. As the P2P network grows we add more machines to scale with the number of incoming mapping requests and monitoring functionality.

Available nodes send periodic UDP messages every 10 seconds to report on their status information to the DTS machines. When DTS does not hear from a node for 20 seconds it no longer includes it in the pool of available nodes. These messages are very light and are easily handled by the DTS.

When the network grows very large the participating nodes are split into separate pools and each pool is assigned a DTS set. The DTS machines from that set assign nodes only from their pool. Each shared file is also assigned to a specific DTS set. In this way, for a given content file a single DTS set can keep track of all the Peers for that content and assign live nodes from the same pool. That ensures consistency of the node maps.

3.4 Downloading Chunks

Before downloading data a Peer A must discover which Inode will be able to get that data. Not every Inode that it talks to will lead it to the data because 1)the Peers that the Inode talks to may not have the chunk that the Peer A wants and 2)the Inodes are restricted from serving certain chunks. Below we present the discovery and download heuristics used by the nodes.

3.4.1 Discovery Heuristic

A Peer maintains TCP connections to all of the live nodes in its node-map. A Peer marks each node in the map-set as "busy" , meaning that it's in process of downloading chunks on behalf of the Peer or "free". If one such node becomes "free" the Peer attempts to discover and then download a subset of chunks via that node.

For discovery the Peer simply picks a random subset of the chunks that it's still missing and sends a discovery query to the selected Inode. (The query contains the content name and the chunk numbers) The Inode then re-broadcasts that request up the download graph. Any intermediate node or

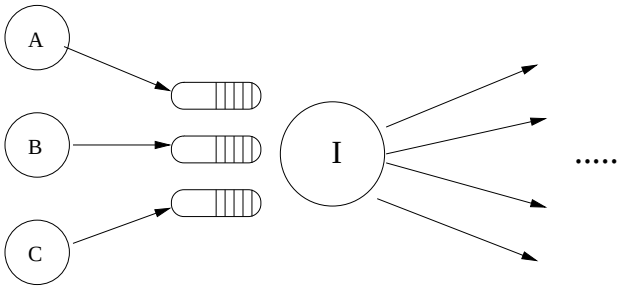

Figure 4: Request queues. An intermediate node I queues requests from upstream nodes A, B, and C, and services requests via round-robin of the queues

the Peer that receives the query replies with the chunks from the requested subset that it does have. The replies are then propagated through the tree to the requesting Peer.

We add one important feature that helps starting Peers share their chunks efficiently. If a Peer that receives a query request does not have all of the requested chunks it adds additional randomly selected chunks that it does have. This feature then allows new Peers to share the little data that they do have making the system more efficient.

Although we use broadcast for discovery requests, the overhead is quite small as the Inode's query is only broadcast to two extra levels. In case of a very large degree D instead of broadcast the discovery proceeds via a randomly selected branches and only if the chunks are not discovered other branches are attempted.

3.4.2 Download Heuristic

Once the Peer receives a reply to its query from the selected Inode it chooses a subset of the discovered chunks to be downloaded via that Inode. It sends a download request for the chunks together with the cash coins to pay for the download service. This node is marked "busy" by the Peer until it finishes downloading all the requested chunks or the Inode replies saying that it cannot complete the download of some of the requested chunks. Once the Inode becomes "free" the process of discovery and download over that Inode repeats.

The Inodes themselves use the same heuristics to discover and download chunks. Each Inode maintains a list of requested chunks from each upstream node. For example, figure 4 shows Inode I with queues of download requests from each of the nodes A, B and C. In order not to starve any particular queue, the Inode round-robins through the three queues and picks chunks from each for discovery and download.

3.4.3 Downloading Missing Chunks

It is likely that a Peer will not be able to download a small number of the required chunks via its inodes. This is partially due to the chunk-map restrictions placed on the Inodes. In addition, a Peer may loose some electronic coins that it needs to pay for downloads due to some nodes going down before they completed an upload of chunks already paid for by that Peer.

Towards the end of the download process when the Peer can no longer get chunks it connects to the content provider to download the missing chunks. As will be shown in section 4 the fraction of the file that the Peer is missing should be very small (less than 1%) and it does not require much resources from the content provider to handle this download. However, such downloads are throttled to discourage misbehaving nodes from using the content provider directly.

Also, before completing the download, the content provider invalidates all of the uncashed coins associated with the Peers's purchase of the content. This prevents a Peer from saving all of its coins for later use.

3.5 Reputation

Aequitas uses a reputation mechanism to deal with some forms of maliciousness on behalf of the nodes. In particular, this mechanism deals with a case of a node that makes a request that does not match its certificate or it cashes the coins and fails to produce data. A node that detects such behavior reports the node id of the mis-behaving node to the DTS. After k reports from distinct nodes of such behavior over a small period of time the mis-behaving node is punished by being taken out of the system for days. The node cannot earn cash during that period. Files purchased by this node cost extra cash and are downloaded via a throttled connection from the content provider.

Nodes that makes illegal requests is clearly being malicious. We set the value of k very small for that case (e.g. k=3) just to make sure that it's reported by at least a few distinct nodes. When a node cashes the coins but does not upload the chunks is treated more leniently as such a node may simply be malfunctioning. We set $k = D/2$ to cover at least half of its neigbors.

To prevent users from running multiple cash-stealing nodes we require users to register with the system and make a minimal deposit of $1. Each time the user joins the system they must login securely to obtain a signed time-limited token that they must present when establishing connections to other nodes.

4 Analysis

We now analyze some of the properties of the system stated in the introduction. We will formally analyze the probabilities that certain bad events can happen and will then show, that for typical scenarios, these event are extremely unlikely. The three properties we analyze are that anonymity is preserved, that only authorized downloads occur and that the load on the content provider is not too heavy. A fourth property of our system is that nodes are compensated proportionately for their bandwidth. We do not present the analysis of this property as it follows directly because in this system, nodes first cash their coins and then upload the data.

4.1 Anonymity

Due to the anonymity imposed by the intermediate nodes, Peers in a given file-sharing instance are highly unlikely to learn of one another's identities and collaborate in a way that may hurt the content provider's interests. Consider what must happen in order for two Peers to collude. Referring to figure 1, in order for Peers A and D to collude they both must want to collude and there must be at least one path between A and D where each Inode on the path is willing to identify its successor and predecessor.

Let's now focus on one particular Inode, say A, and calculate the probability that it will find another Inode with which to collude. (We abbreviate this event as "A learns".) Assume that each Peer wants to collude with probability p and that the probability an Inode wants to collude is i. For a particular path from A to another Inode, the probability that all three nodes on the path are willing to collaborate is i^2p. We call a path in which all three nodes collaborate *bad*, otherwise we call the path *good*. Standard calculations now state that

$$Pr(\text{A learns}) = 1 - Pr(\text{every path from A to anther Inode is good})$$

By construction of the download graph (section 3.3.1) each Peer talks to D I1 Inodes, each I1 Inode talks to D I2 Inodes and each I2 Inode talks to exactly one Peer, and thus there are at most D^2 paths. (We say at most because that paths need not all be distinct).

Thus

$$\begin{aligned} Pr(\text{A learns}) &= 1 - (1 - i^2p)^{D^2} \\ &\approx 1 - e^{-D^2 i^2 p}. \end{aligned}$$

Let's assume a realistic file-sharing instance with out-degree $D = 20$ and where each paying Peer has a probability of $p = 1/10$ desire to participate in a malicious collusion and the probability an Inode wants to participate is $i = 1/40$. (An Inode has little to gain from such collusion and thus we assume $i << p$). Then the probability that A learns is approximately .025.

Now, by our model, A itself is only malicious with probability $1/10$, and so given a node, the probability that it is has both motive and opportunity to collude is only .0025. Now, we can use standard calculations to conclude that we can have up to 200 nodes before the expected number of compromised identities is $1/2$, and 400 nodes before it is 1. Now using Chernoff Bounds, one can show that the probability that there is a large number of compromised nodes is extremely unlikely.

4.2 Authorized Download

In this subsection, we show that a set of colluding Inodes is unlikely to obtain a full copy of the content that they are helping distribute. Recall that nodes that serve as intermediates are not allowed to serve full files, but rather they are assigned a random fraction of chunks. We use these facts in the following analysis.

Let k be the number of colluding nodes, and recall that the Inodes are chosen to serve given content from a large pool of nodes, with some probability t. The expected number of colluding nodes assigned to serve one file is thus kt. Each of these Inodes is assigned to serve a fraction f of the n data chunks. Consider a particular chunk, and observe that the probability that a given chunk is not found among any of these nodes is $(1 - f)^{kt}$. Thus we conclude the following:

Lemma 4.1 *If k colluding nodes each are chosen to serve content with probability t and has a fraction f of n data chunks, then the expected number of chunks they will be missing is $n * (1 - f)^{kt}$.*

Assuming $k = 50$, $p = 1/10$, $f = 0.4$ and $n = 5000$ then the expected number of chunks they will be missing is 389. Using Chernoff Bounds it's easy to show that the probability that they are missing fewer than 200 chunks is less than 10^{-10}.

4.3 Small Load on Content Provider

Using analysis very similar to above we can show that the number of missing chunks that a Peer will not be able to find among its Inodes is small. Thus, the total fraction of missing chunks that it will need to get from the Content Provider is small.

By construction of the download graph a Peer and each I1 node have a degree D. Using the same f and n as defined above it's easy to see that the probability that a Peer can not find a chunk among it's I1 nodes is $(1 - f * (1 - (1 - f)^D))^D$. (It would be just $(1 - f)^D$ if we don't count the possibility that an I1 cannot find a chunk among it's I2s). The expected number of missing chunks is thus $n * (1 - f * (1 - (1 - f)^D))^D$. Taking $D = 20$, $f = 0.4$ and $n = 5000$ we get the expected number of missing chunks to be less than 0.2. To account for a possibility that half of the nodes are dead or are slow we can equivalently assume the degree of $D = 10$. This would still mean that we expect only 0.5% of the chunks to be downloaded from the content provider.

5 Evaluation

5.1 Setup

In order to evaluate the performance of Aequitas we implemented the required components: P2P nodes, DTS, and the Bank/Content Provider. The P2P nodes and the DTS where implemented in C and use non-blocking TCP connections for communication. The Bank consists of an Apache front-end connected to a MySql database back-end. MySql is used to keep track of signed coins and the users' balances. The logic to handle node requests for bank transactions is handled with PHP modules.

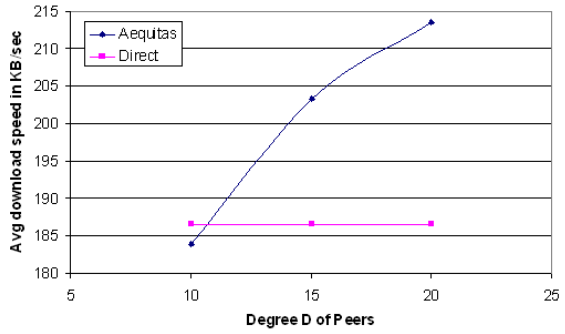

Figure 5: Aequitas as graph outdegree changes vs "direct download" speed

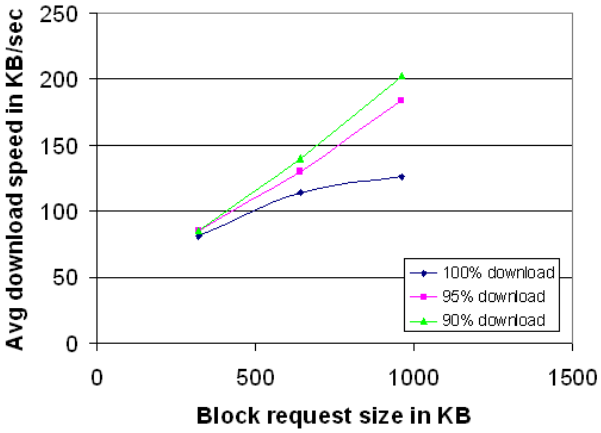

Figure 6: Average download times vs. request size

All of our experiments were run on a testbed of 400 PlanetLab [25] nodes. These are all Linux Boxes that run a 2.6 Linux Kernel.

We use this testbed to test and tune the performance of Aequitas by varying various parameters of the system such as the size of the request, the degree of the download graph and a number of scheduling heuristics.

All of the file-sharing instances we experiment with involve randomly picked Peers (about 40). The rest of the nodes form the pool of available Inodes. We always use the same 3 Seed nodes (i.e. nodes initialized by the content provider to have all the chunks of the files). For all of our tests we use a dynamic model where file-sharing Peers enter the system with uniform arrival times. This model of Peers joining at periodic intervals more accurately models real-world situations than having all Peers join simultaneously. The average arrival time between subsequent Peers in our tests is 10 seconds, so it takes roughly 400 seconds for all 40 Peers to join the system. A typical file size is 44MBytes, which is comparable to the size of a long music video or a short TV show episode.

5.2 Basic Parameters

In our first graph (Figure 5), we compare the performance of Aequitas to the "direct download" model where the Peers download directly from the Seeds managed by the Content Provider. For this test we picked 40 random US and non-US Peer nodes to download the 44MB File. We experiment by varying the degree D of the download graph (the outdegree of the Peer and I1 nodes as described in section 3.3.1). The y-axis measures the average download speed in KBytes/sec of the Peers downloading this file.

As we increase the degree D the Peers have more Inodes to request their files from. They are more likely to find faster Inodes and thus the average download speed increases. Only for $D = 10$ does the direct download slightly outperform Aequitas. For $D = 20$ the average download speed of Aequitas Peers outperforms "direct download" by more than 20KBytes/sec.

We next look at how the request size effects the average download speed. When making a download request, nodes request several chunks at a time (as described in section 3.4). The request size is the number of chunks times the size of a chunk. Intuitively bigger request sizes are more efficient because they leave less idle time between requests. It's especially true with a system like Aequitas, in which requests travel up and down layers of nodes. This relationship between request size and average download rates is captured in figure 6. The figure has three curves for the average download rates of 90%, 95% and 100% of a file. (That is, the 90% curve shows the average rate at which the Peers download the first 90% of the file.). For the 90% curve, requesting 1MB at a time almost quadruples the download rate (from 320KB requests). For the 100% curve the effect of larger request size doubles the rate. The effect is still significant but not as pronounced because towards the end of the file download there is a long tail where the Peer may be waiting for chunks that it requested from a slow Inode. We next look at the technique called boosting that helps mitigate slow tail download.

5.3 Boosting

As shown in figure 6 above the tail of the download time can be quite long. This happens when a Peer has all the chunks but is waiting for some download to complete from one or two very slow Inodes. To deal with slow tail downloads we implement a technique we call boosting. Boosting simply means that when the client has downloaded all but a small number of remaining chunks and the rate of download is very slow it simply takes the chunks and re-requests them via other Inodes.

Figure 7 measures the same relationship between request size and the download rate, but now using boosting. This graph compares the average download rate for downloading the entire file with and without boosting. One can see that the 100% average download times improves by about 20 KBytes/sec with this technique.

To determine when the download enters the boosting mode, we watch the download rate over the moving 15 sec-

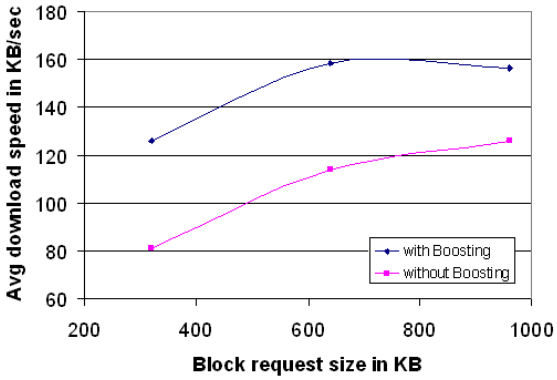

Figure 7: Improvement of download rate due to boosting

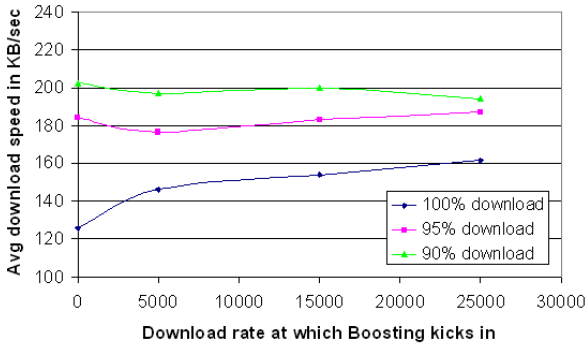

Figure 8: Average download rate vs. the download threshold at which boosting kicks in

ond window. If the download rate falls beyond a certain threshold the boosting mode begins. Figure 8 examines the improvement in the average download of the file for the thresholds of 5KBytes , 15KBytes and 25KBytes/sec. (The point 0 on the x-axis represents the case where Boosting is not used). As expected the higher the threshold the faster the speedup for the average download of the file (lower curve). This occurs because, at higher threshold, we don't wait for the download rate to drop as much before switching into boosting. Unsurprisingly, the rate of the download of the first 90% of the file does not depend much on boosting.

Of course using boosing requires paying extra coins to the Inodes since we repeat the requests already sent to the slower nodes. In our evaluation we found that the improvement due to boosting raised the number of coins required to pay for extra downloads only by 4-6%. (The total cost in coins for a file download is already only a fraction of the file cost).

5.4 Variations in Other Parameters

We now look at the effect of downloading multiple files and the effect of some nodes dying or unexpectedly leaving the system. We continue to use boosting for these measurements.

We also experiment using our file-sharing network for simultaneous download of 3 44MB files. Each file in this in-

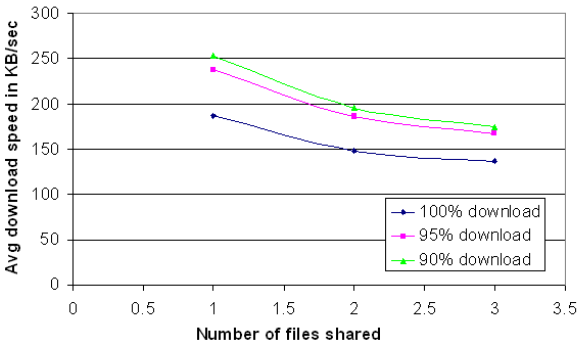

Figure 9: Multiple files shared over the same P2P system

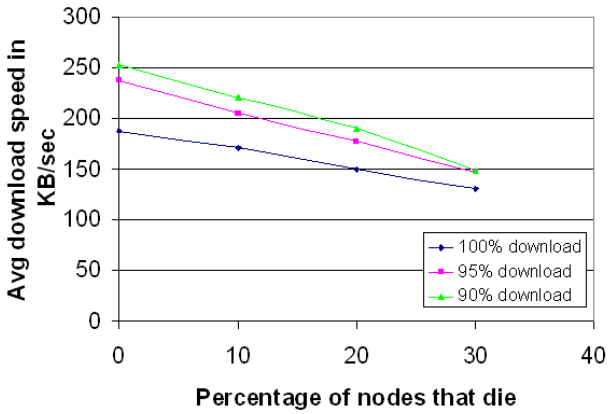

Figure 10: The effect of killing a percentage of nodes duing the download

stance is being download by 30 clients. Figure 9 shows the average download times experienced by all clients in each instance (i.e when one file is being downloaded when 2, and when 3 are downloaded). The graph shows that the average download rate does drop slightly with the increased number of files as the system bandwidth becomes more saturated. The drop is sublinear however as more aggregate bandwidth is used with more files.

We next examine how the system copes as some nodes become unavailable. We generate a schedule for killing a fraction of the nodes on the network after the download process has begun and measure the average download speeds of the Peers. Figure 10 shows the average download speed as we schedule to kill off 10%, 20% and 30% of randomly picked P2P nods. As some nodes join the network dynamically other nodes are killed off at the rates of 10, 20, and 30% relative to the joining rate. As one would expect the reduction in the average download speed is proportional to the rate at which nodes are being killed off. For the 30% rate of killing nodes the download rate drops by about 25%.

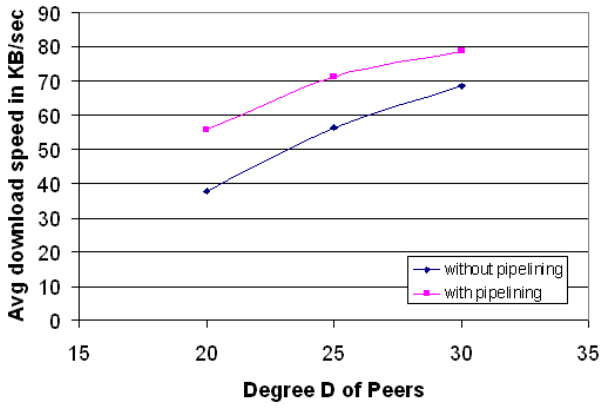

Figure 11: The effect of pipelining and degree on download times

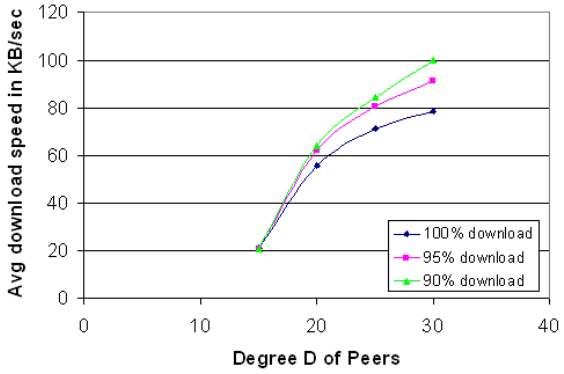

Figure 12: The effect of chunks constraints on download times

5.5 Chunk Restrictions and Pipelining

We next perfom tests in which we only allow Inodes to download randomly assigned chunks. The heuristics required to schedule download requests in this scenario are more complex than those without restrictions. For instance, there is a high probability that a small number of requests received by an I1 Inode may only be served by one of its I2 nodes. If that I2 node is busy the requests will block causing slow downs in the system.

To deal with this scenario we implemented another technique called Pipelining where we allow nodes to add to the request queues of their upstream nodes even if those queues are not empty. So even if a queue is blocked because it is hard to find an upstream node that can service this request, it can still receive more requests from its downstream node without blocking the system. Figure 11 shows the improvement due to pipelining when we use chunks restrictions on the node. The performance is improved with both the use of pipelining and the higher out-degree which increases the probability of finding a node which can serve a given chunk. Figure 12 demonstrates the download times for the first 90%, 95% and 100% of the file respectively, using pipelining.

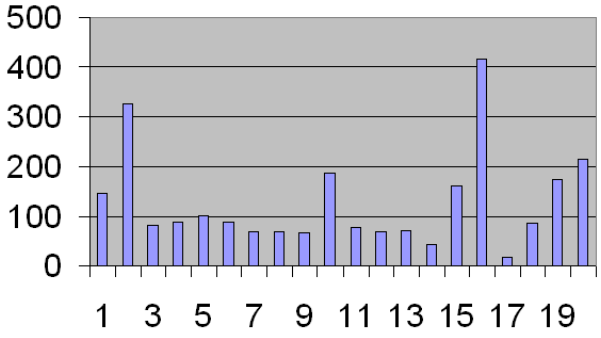

Figure 13: Number of missing chunks from random 10 Inode samples

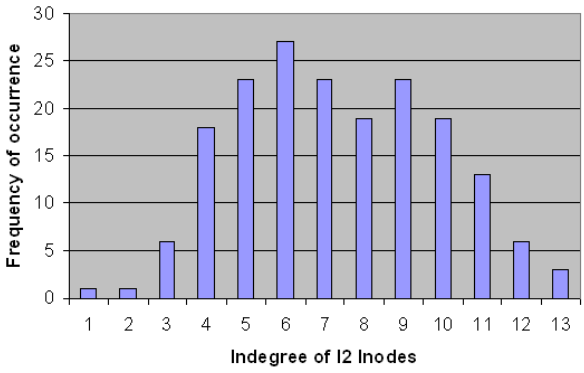

Figure 14: Indegree of I2 Inodes

5.6 Aequitas Mapping Properties

Figure 13 demonstrates that even if a significant subset of the Inodes are restricted to serving random fractions of the same file, they still cannot combine to obtain a full file. Here each Inode is assigned 40% of the total chunks. We collected 10 random samples of 10 Inodes each to show that each set of nodes combined is still missing parts of the file.

Figure 14 demonstrates the indegree of an I2 node. The normal curve shows the stability of the consistent hashing function that maps I1 nodes to I2s.

The properties of consistent hashing guarantee that the mapping of nodes will change only slightly despite nodes joining and leaving the network. For instance Figure 15 shows the remapping of a number of Peers for a graph with outdegree 5 when 10% of Inodes leave the system. Most of the Peers remain mapped to 5 Inodes, while a few are mapped to one or two new nodes.

6 Conclusion and Future Work

We have tackled an important problem in the content distribution industry: leveraging P2P file sharing for legitimate content distributions. To address this problem we brought together several concepts including anonymity and the fair use of economic incentives. Although these concepts are not

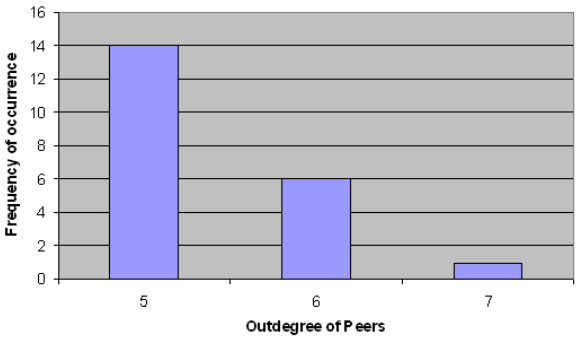

Figure 15: Remapping of Peers with outdegree 5 due to network changes

new, we use them in a novel and inter-dependent way, and show that they do not compromise anonymity. Furthermore, our system carries a number of beneficial side-effects, such as privacy (that comes with anonymity) and fair use of resources (due to direct payments). The incentives also convince users that finished the download to stay longer online and continue speeding up downloads for others.

We have shown in evaluation that the extra layers of anonymity do not add extra overhead to the "direct download" methods. In fact, as we increase the degree of the download graph (the number of nodes that a Peer talks to) we achieve more than 10% improvement over a direct download system. We have introduced techniques such as boosting and pipelining that help the system heal and continue perform efficiently despite the existing bottlenecks. In particular, Boosting helps improve the slow tail of the download common in P2P systems.

We see much potential in the future work on this system in areas including economic incentives, reputation building, DRM and efficient tuning. Although in is clear that incentivized Peers that stay online longer help improve performance for other Peers, it would be interesting to see how much longer would the Peers remain on line. We plan to conduct a user study with the prototype that we have created to study those effects.

Moreover we believe that adding intermediate nodes not only helps with anonymous communication but will, with further system tuning, enhance performance efficiency. The existence of intermediate nodes creates more available total download bandwidth as well as more routing paths to route around potential glitches between communicating Peers.

Another area of future work is to use a fluid economic model rather then fixed prices. Intermediate nodes and Peers could bid for service in response to discovery messages. The bidding has potential of reducing the infrastructure costs for the end-user even further. We will then evaluate whether such a bidding system will improve system performance.

References

[1] J. G. Aguilar. personal communication, February 2006.

[2] Akamai. http://www.akamai.com/.

[3] Apple Itunes. http://www.apple.com/itunes.

[4] K. Bennett and C. Grothoff. GAP — practical anonymous networking, 2003.

[5] Bit Torrent. http://www.bitconjurer.org/bittorrent.

[6] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. In *Proceedings of ACM*, pages 84–90, February 1981.

[7] A. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 311–320, July 2000.

[8] L. P. Cox and B. D. Noble. Samsara: Honor Among Thieves in Peer-to-Peer Storage. In *SOSP '03*, 2003.

[9] D.Goldschlag, M.Reed, and P. Syverson. Onion Routing for Anonymous and Private Internet Connections. In *Communications of the ACM, vol. 42, num. 2*, Feb 1999.

[10] R. Dingledine, N. Mathewson, and P. Syverson. Reputation In P2P Anonymity Systems. In *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[11] R. Dingledine, N. Mathewson, and P. Syverson. TOR: The second generation onion router. In *Usenix Security*, 2004.

[12] R. Dingledine and P. Syverson. Reliable mix cascade networks through repuation. In *In Proc. Sixth International Financial Cryptography Conference - FC02*, 2002.

[13] P. Druschel and A. Rowstron. PASTRY: Scalable Distributed object location and routine for large-scale peer-to-peer systems. In *IFIPS/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[14] D. R. Figueiredo, J. K. Shapiro, and D. Towsley. Payment-based incentives for anonymous peer-to-peer systems. Technical report, July 2004.

[15] E. Franz, A. Jerichow, and G. Wicke. A Payment Scheme for Mixes providing anonymity. In *Proc. Trends in Distributed Systems for Electronic Commerce (TREC '98)*, volume 1402, pages 94–108. Springer-Verlag, 1998.

[16] M. Freedman and R. Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of ACM Conf. On Computer and Communications Security (CCS)*, November 2002.

[17] G.Denezis, R.Dingledine, and N.Mathewson. Mixminion: Design of type iii anonymous remailer protocol. In *IEEE Symposium on Security and Privacy. IEEE CS*, pages 12 – 15, May 2003.

[18] Gnutella. http://rfc-gnutella.sourceforge.net.

[19] I2P. http://www.i2p.net.

[20] D. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *29th Annual ACM Symposium on Theory of Computing*, pages 654–663, May 1997.

[21] Kazaa. http://www.kazaa.com.

[22] U. Moller, L. Cottrell, P. Palfrader, and L. Sassman. Mixmaster protocol http://www.abditum.com/mixmaster-spec.txt, July 2003.

[23] A. Nandan, G. Pau, and P. Salomoni. Ghostshare - reliable and anonymous p2p video distribution.

[24] Napster. http://www.napster.com.

[25] PlantLab. http://www.planetlab.org/.

[26] T. Poutanen, H. Hinton, and M. Stumm. Netcents: A Lightweight Protocol for Secure Micropayments. In *Proceedings of the 3rd USENIX Workshop on Electronic Commerce*, 1998.

[27] Prodigem. http://www.prodigem.com.

[28] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scaleable Content-addressable network. In *ACM SIGCOMM*, 2001.

[29] M. Reiter and A. Rubin. Crowds: Anonymity for web transactions. In *ACM Transactions on Information and System Security*, volume 1, pages 66–92, 1998.

[30] M. Rennhard and B. Plattner. Practical anonymity for the masses with morphmix. In *Financial Cryptography, Springer-Verlag, LNCS*, 2004.

[31] R. Rivest and A. Shamir. Payword and micromint - two simple micropayment schemes. In *International Workshop on Security Protocols*, volume 1189, pages 69–87. Springer-Verlag, 1996.

[32] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. CHORD: A scalable p2p lookup service for internet applications. In *ACM SIGCOMM 2001*, August 2001.

[33] R. S. Tracker. http://www.inkrecharge.com/ttrc2/.

[34] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer. Karma: A virtual Currency for Peer-To-Peer Systems. In *ACM Workshop on the Economics of Peer-to-Peer Systems*, June 2003.

[35] Xbox-sky. http://bt.xbox-sky.com/.

[36] B. Yang and H. Garcia-Molina. PPay: micropayment for peer-to-peer systems. In *ACM Conference on Computer and Communication Security (CCS)*, volume 10, pages 300–310. ACM Press, 2003.

[37] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location. Technical report, April 2001.